C-016

マイクロプロセッサアーキテクチャツール MEIMAT の開発
- 汎用命令の提案 -
Development of Microprocessor Architecture Tools, MEIMAT
- Proposal of Meta-instruction -

榎本 崇† 花井 北斗† 鈴木 祐一郎† 佐野 友輝† 森本 智之†,‡ 堤 利幸†
Takashi Enomoto Hokuto Hanai Yuichiro Suzuki Yuki Sano Tomoyuki Morimoto Toshiyuki Tsutsumi

1. 研究概要

MEIMAT(Meiji University Microprocessor Architecture Tools)は明治大学で開発している任意のマイクロプロセッサを設計するためのツールである。命令の動作を、その命令で使用されるハードウェアモジュール(ハードウェアの部品)間の接続情報のブロック図として可視化させることができる。MEIMAT では命令を、意味を中心とした側面と機能を中心とした側面の 2 つの面から命令を考えることで、ユーザが命令の持つ意味とその動作を理解することができる。その結果ユーザは命令設計レベルからマイクロプロセッサを設計することが可能になる。今回の研究では多様な命令設計に対応するためにどのようなプロセッサの命令でも表現することができる汎用命令の仕様を提案する。

2. 研究背景

マイクロプロセッサの設計ツールは図 1 に示すようにハードウェア記述言語や論理合成ツールを使用した論理設計レベルのものが多い。アーキテクチャレベルで使用を探索するツールや動作合成を行うツールもでてきている[1][2]。また、論理設計をベースとした設計教育ツールなども提案されている[3][4]。しかしながら、これらのツールは設計者自身がプロセッサの仕様そのものを考えずとも設計できてしまうため、マイクロプロセッサの仕様そのものを設計する力が身につかないという問題があった[5]。

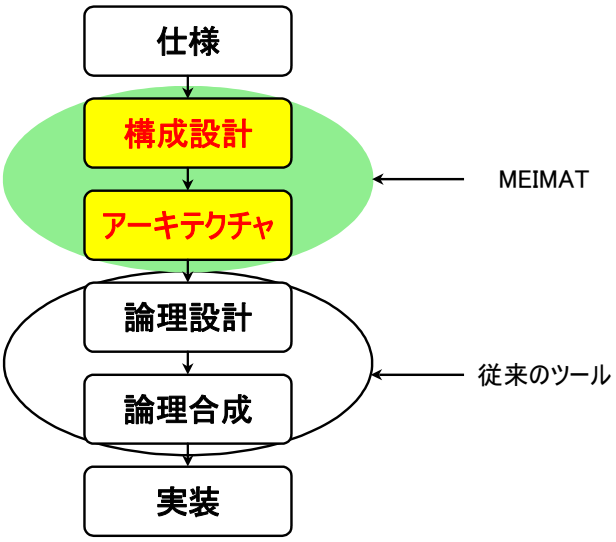

図 1 設計ツールのレベル

3. CASL II 命令対応の MEIMAT の開発

MEIMAT は図 1 に示すようにアーキテクチャレベルでのマイクロプロセッサの設計を目指して行われている研究である。我々はアーキテクチャレベルでプロセッサの仕様を考えるためには、命令の意味とプロセッサのハードウェアの対応が重要であると考えている[6]。プロセッサを設計するためには、命令を実現するために必要なハードウェアが何かわからずに設計することはできないためである。このような考えから、我々の研究室は命令の意味とプロセッサのハードウェアの対応をわかりやすく理解させることができる命令ダイアグラムツールを開発した[7]。

命令ダイアグラムツールは情報処理技術者試験で使用される COMET II プロセッサで用いる命令セット CASL II を用いて、各命令においてプロセッサのハードウェアがどのように動作しているかをブロック図として図示できるツールである。命令のブロック図を図示することにより、命令の意味と使用するハードウェアの対応が視覚的に理解できるので、プロセッサの仕様を考えることができる。

しかしながら、命令ダイアグラムツールは CASL II 命令の表記をそのまま使用しているため、COMET II にしか対応できないという問題があった。任意のマイクロプロセッサの設計を行うためにはどのような命令でも設計できるようにする必要があり、そのためには設計できる命令を汎用化する必要がある。

4. 汎用命令対応の MEIMAT の開発

MEIMAT は命令ダイアグラムツール[7]で実現することができた、命令の意味と使用するハードウェアの対応の理解を、あらゆるプロセッサでも実現することを目指している。そのためにはどのようなプロセッサの命令でも表現することができる汎用命令と、どのようなハードウェアでもパイプラインステージごとにハードウェアモジュールがどのように接続されているか図示できるツールの開発が必要である。また、ユーザが定義したプロセッサがどのような仕様なのか判断するためには実際にプロセッサを動作させる必要がある。そのためにプロセッサを動作させその性能を評価するシミュレータの開発を目指している。本論文は汎用命令の基本形を定義し、COMET II において汎用命令を用いた命令ダイアグラムツールが開発できたので報告する。

5. 汎用命令の提案

汎用命令は意味を中心とした意味分割表記とハードウェアを中心とした機能分割表記により構成される。従来までは意味分割表記を CASL II 命令で表記していた[7]が、汎用命令では我々が考案した記述法で表記する。また、機能分割表記は従来までの機能分割表記[7]をベースに改良した、

†明治大学
‡株式会社ヴィアックス

パイプラインステージにおけるハードウェアモジュールとその接続情報を中心とした表記となる。

5.1 汎用命令の意味分割表記

アセンブリ命令を 1 つずつ見てみると、それぞれの命令は対象となるデータ（オペランド）とオペランドに対する処理（オペコード）からなっている。命令の処理は算術演算や論理演算など 2 項演算またはデータの移動に関する処理と、プロセッサの SR(ステータスレジスタ)をもとに条件を判断して実行する処理が多い。このような理由から意味分割表記は 2 項演算をベースに SR の条件の判断と SR を保持するかを判断する記述で構成されている。

意味分割表記は表 1 に示す要素から構成され、表 2 に示す仕様に従い 1 行で記述する。CASL II のリターン命令のように命令の処理を表現するのに複数の表記が必要な場合は、1 行に意味分割表記を処理の順番ごと記述する。記述の統一を図るため、必ず 2 項演算は右辺の結果を左辺に代入するという記述に統一している。それぞれの要素については 5.1.1 以降で詳しく説明する。

表 1　意味分割表記の要素

要素	要素の意味
prestatus	SR のチェック
destination	書き込み先
source1, source2	ソース
@module=operation	演算、オペレーション
poststatus	SR の更新

表 2　意味分割表記記述の仕様

```
prestatus, (destination)<-(source1)@modul(source2),
@modul=operation, poststatus;
```

5.1.1 prestatus

prestatus は命令を実行する前に SR を参照し、命令を実行するかどうかを判断する部分となっている。表 4 に prestatus の実行条件を示す。また、SRal(無条件実行）の時は省略可能である。例えば、SRmi と記述していた場合は SR の中身をチェックし、SR の中身がマイナスなら命令を実行するという意味である。

表 4　prestatus の実行条件

実行条件	実行条件の意味
SRal	SR always
SRpl	SR if positive or zero
SRmi	SR if negative
SReq	SR if equal
SRne	SR if not equal
SRvs	SR if overflow
SRvc	SR if no overflow
SRhs	SR if unsigned higher or same
SRlo	SR if unsigned lower
SRhi	SR if unsigned higher
SRls	SR if unsigned lower or same
SRge	SR if greater than or equal
SRlt	SR if less than
SRgt	SR if greater than
SRle	SR if less than or equal

5.1.2 destination

destination は命令の結果の書き込み先を指定する。命令の結果が書き込まれるレジスタやアドレスを表す。レジスタやメモリの場合は“( )”を付けると、そのレジスタやメモリの内容を指す。書き込み先は汎用レジスタ“(GRd)”やスタックポインタ“(SP)”などがある。メモリの値はアドレスを表す“ADRd”を使用し、“MEM((GRd)+ADRd)”と表記する。また、分岐命令などは PC(プログラムカウンタ)に対する代入と考えることができるので destination に“(PC)”を記述する。比較命令など、処理の結果を書き込まない場合は destination に“nil”を記述する。

5.1.3 source1, source2

source1 と source2 は命令の処理に使用するデータを指定する。演算に使用するレジスタやアドレスを表す。データ移動命令の際は source2 は存在しない。destination と同様、“( )”を付けるとそのレジスタ、メモリの内容を指す。汎用レジスタの値は“(GRs1)”と表記し、メモリの値はアドレスを表す“ADRs”を使用し“MEM((GRs1)+ADRs1)”と表記する。また、SP と PC は destination と同様に“(SP)”、“(PC)”と表記する。

5.1.4 @module=operation

@module=operation は 2 項演算で実施する演算を定義する。演算については複数の演算ユニットを使用する場合が想定されるため、命令に使用する演算器(module)とオペレーション(operation)を示す。COMET II の場合は演算モジュールは ALU しかないので、module は ALU のみとなる。また、データ転送命令など演算モジュールを使用しない際は記述しない。

5.1.5 poststatus

poststatus は命令を実行した後、プロセッサの SR を変更するか定義する部分である。SR をセットするときは SRset、しない時は SRunset と記述する。SRunset の時は表記上省略可能である。

5.1.6 意味分割表記の例

CASL II 命令を汎用命令の意味分割表記に変換した場合の例を表 5 に示す。

表 5　意味分割表記の例

例 1; 算術命令	ADDA GRn, GRm ⇩ (GRd)<-(GRs1)@ALU(GRs2), @ALU=ADD, SRset;
例 2; データ 移動命令	LD GRn,adr,GRx ⇩ (GRd)<-Mem((GRs1)+ADRs1);
例 3; ジャンプ 命令	JMI adr,GRx ⇩ SRmi,(PC)<-(GRs1)+ADRs1,@ALU=nil,SRunset;
例 4; 比較命令	CPA GRn,GRm ⇩ nil<-(GRs1)@ALU(GRs2), @ALU=SUB, SRset;
例 5; リターン 命令	RET ⇩ (PC)<-Mem((SP));(SP)<-(SP)+1;

5.2 汎用命令の機能分割表記

プロセッサの制御回路がパイプラインステージごとにどんなハードウェアモジュールを使用し、そのハードウェアモジュールの処理結果をどのハードウェアモジュールに受け渡すかと言うことがわかれば、プロセッサの 1 命令を実現することができる。この理由から、機能分割表記はパイプラインステージにおけるハードウェアモジュールの接続情報で構成される。表 6 の示しているパイプラインステージは従来の機能分割表記と同じステージを想定している。従来の命令ダイアグラムツールで使用していた機能分割表記はパイプラインステージの順番と図 2 に示すハードウェアモジュールのポート間の接続情報だけを記述していた。表 7 に示す今回の機能分割表記の仕様は、よりユーザが理解しやすくなるように、どのパイプラインステージのハードウェアモジュールのポートなのか簡単にわかるようになっている。機能分割表記の仕様は 2 つの部分に分かれる。前半部は 1 行ずつパイプラインステージをステージ順に記述し、そのステージで使用されるハードウェアモジュールを記述する。後半部はハードウェアモジュール同士の 1 対の接続を一行ずつ記述する。表 8 は仕様に従い CASL II の加算命令(ADDA)を機能分割表記で記述したものである。

表 6 パイプラインステージ

ステージ名	ステージの役割
IF(Instruction Fetch)	命令読み込み
ID(Instruction Decode)	命令解読
OF1(Operand Fetch1)	オペランド読み込み
OF2(Operand Fetch2)	アドレス計算
MEM(Memory access)	メモリアクセス
ALU(Arithmetic Logic Unit)	演算機能
WB(Write Back)	ライトバック

表 7 機能分割表記の仕様

```
1_IF{使用しているハードウェアモジュール}
2_ID{使用しているハードウェアモジュール}
3_OF1{使用しているハードウェアモジュール}
4_OF2{使用しているハードウェアモジュール}
5_MEM{使用しているハードウェアモジュール}
6_ALU{使用しているハードウェアモジュール}
7_WB{使用しているハードウェアモジュール}
Wire_{
(パイプラインステージ . ハードウェアモジュール . 出力ポート) ->
(パイプラインステージ . ハードウェアモジュール . 入力ポート)
 .
 .
 .
};
```

6. 汎用命令に対応させた命令ダイアグラムツール

我々の研究室で開発済みの CASL II 命令を用いた命令ダイアグラムツールを今回の研究で汎用命令に対応させることに成功した。図 3 は改良した命令ダイアグラムツールに CASL II の加算命令を表示させたものを示している。改良された命令ダイアグラムツールには図 4 に示す直接入力ダ

表 8 機能分割表記の例

```
1_IF{PC,PC_ADDER,TERMINAL,I_CACHE};
2_ID{DECODER};
3_OF1{GRs1,GRs2};
4_OF2{};
5_MEM{};
6_ALU{ALU,ALUMODE,SR};
7_WB{GRd};
Wire_{(IF.PC.out)->(IF.PC_ADDER.pc_in),
(IF.PC_ADDER.out)->(IF.PC.in),
(IF.TERMINAL.plus_one)->(IF.PC_ADDER.val_in),
(IF.PC.out)->(IF.I_CACHE.adr),
(IF.I_CACHE.out)->(ID.DECODER.inst_in),
(ID.DECODER.opr_grd)->(WB.GRd.num),
(ID.DECODER.opr_grs1)->(OF1.GRs1.num),
(OF1.GRs1.out)->(ALU.ALU.in1),
(ID.DECODER.opr_grs2)->(OF1.GRs2.num),
(OF1.GRs2.out)->(ALU.ALU.in2),
(ALU.ALU.out)->(WB.GRd.in),
(ALU.ALUMODE.add)->(ALU.ALU.mode),
(ALU.ALU.status)->(ALU.SR.in)};
```

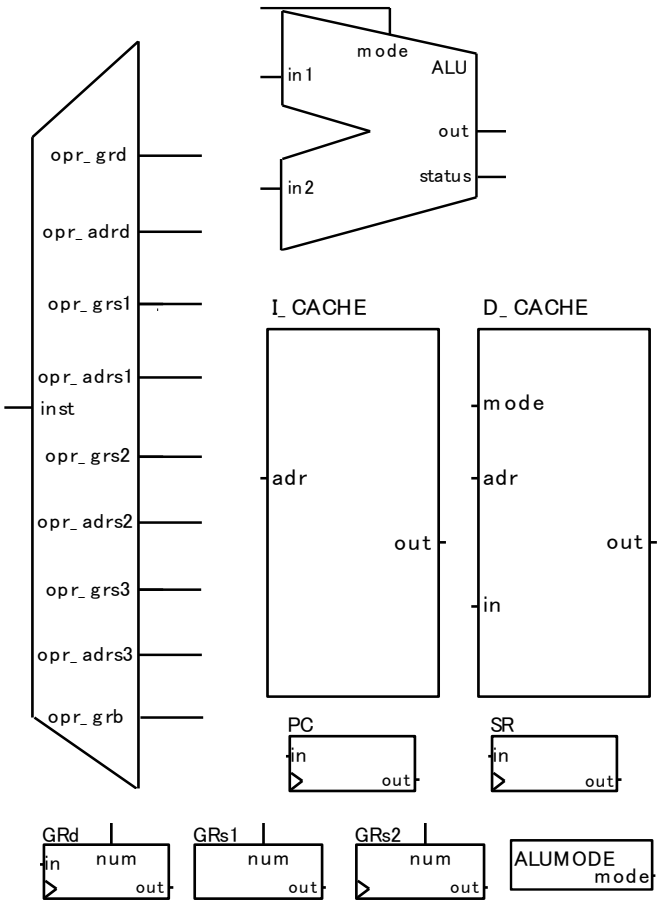

図 2 ハードウェアモジュール

イアログというインターフェースが組み込まれている。直接入力ダイアログでは命令を意味分割表記か機能分割表記で記述することにより命令ダイアグラムツールに記述した命令を表示させることができる。また、直接入力ダイアログでは汎用命令を登録することができ、登録された命令を選択することでその命令を命令ダイアグラムツールに表示させることができる。

7. 結論

我々は汎用命令を提案し、CASL II のすべての命令を汎用命令で表現できることを確認した。また、我々の研究室で開発していた CASL II 命令を用いた命令ダイアグラムツールを汎用命令に対応させることができた。改良した命令ダイアグラムツールではすべての CASL II 命令の表現に成功した。

8. 今後の課題

汎用命令で CASL II 命令の表現に成功した。今後はユーザが定義するあらゆるプロセッサの命令においても汎用命令で表現できるか検証していく。検証には、始めに代表的なプロセッサである MIPS, ARM の命令セットを使用する。また、汎用命令の性能を評価するためにシミュレータを開発して最適な命令セットを探索できるようにすることが今後の課題である。

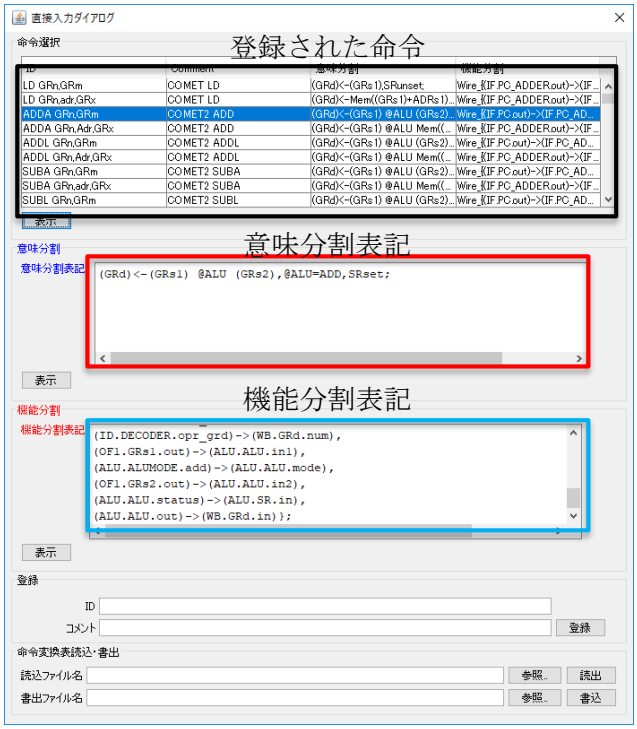

図 4　直接入力ダイアログ

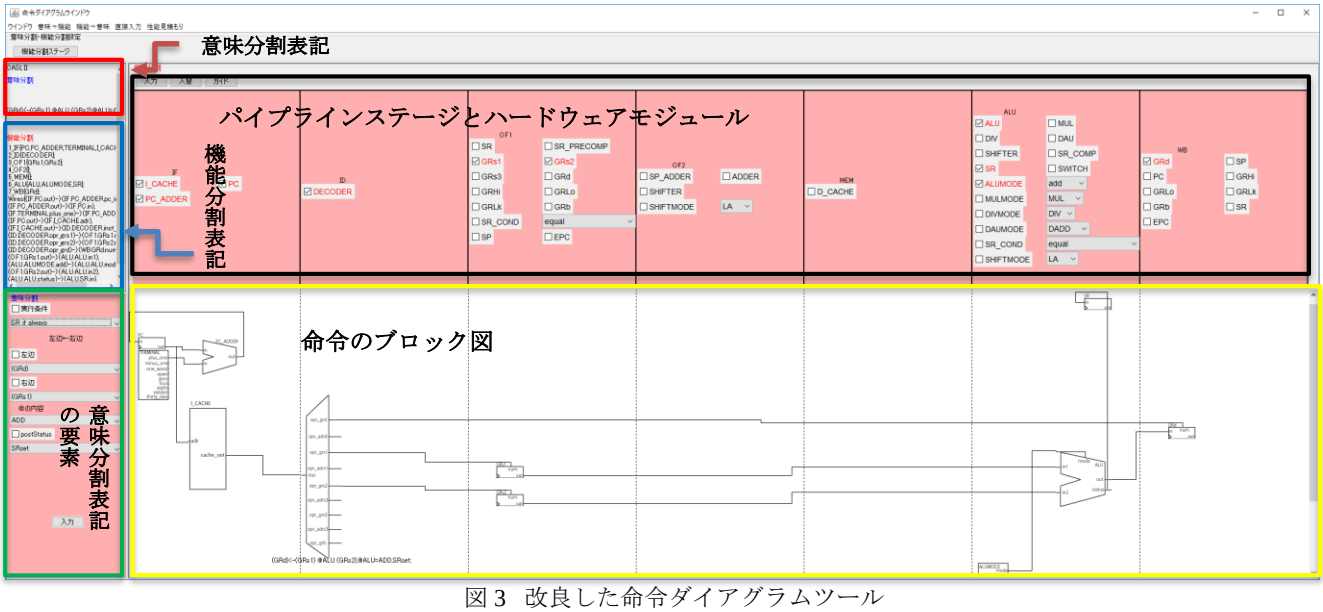

図 3　改良した命令ダイアグラムツール

参考文献

[1] 伊藤真紀子, 塩見彰睦, 佐藤淳, 武内良典, and 今井正治. パイプライン・ハザードを考慮したプロセッサ生成手法の提案. 情報処理学会論文誌, 41(4):851.862, 2000-04-15.

[2] Nakamura Yukihiro, Oguri Kiyoshi, Nagoya Akira, Yukishita Mitsuteru, and Nomura Ryo.High-level synthesis design at ntt systems labs (special issue on synthesis and verificationof hardware design). IEICE transactions on information and systems, 76(9):1047.1054,1993-09-25.

[3] 難波 翔一朗 , 志水 建太 , 山崎 勝弘 , 小柳 滋, "プロセッサ設計支援ツールの設計・実装とハード/ソフト協調学習システムの評価", FIT2007, LC-002, (2007).

[4] 高橋 隆一 , 児島 彰 , 上土井 陽子 , 吉田 典可, "マイクロコンピュータ設計教育環境 City-1" , 情報処理学会研究報告.設計自動化研究会報告 , IPSJ SIG Notes,Vol.97,No.17(19970214)pp 41-48 , (1997).

[5] 森本 智之, 堤 利幸, "パイプラインプロセッサ設計教育システム MEIMES-DESIGN の初期開発", FIT2005, N-025,(2005).

[6] Tomoyuki Morimoto, Toshiyuki Tsutsumi, "Development of Instruction Analysis Tool for Microprocessor DesignEducation", The 17th International Conference on Computers in Education ICCE 2009, pp 489-491, (2009).

[7] 岩本 稜平,森本 智之,堤 利幸 "マイクロプロセッサ設計のための命令ダイアグラムツールの初期開発",FIT2014,C-024(2014)